NASA Technical Memorandum 100553

A SURVEY OF PROVABLY CORRECT FAULT-TOLERANT CLOCK SYNCHRONIZATION TECHNIQUES

Ricky W. Butler

February 1988

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23665-5225

INTRODUCTION

The reliability of a fault-tolerant computer system depends critically upon adequate synchronization between its redundant processors. It is important that the synchronization algorithm maintain proper synchronization of the good clocks even in the presence of other faulty clocks. Many times ad-hoc techniques are used to develop the synchronization system of a fault-tolerant system. Unfortunately, synchronization systems can appear to be sound under a careful Failure Modes and Effects Analysis (FMEA), yet be susceptible to subtle failures. For example, the "intuitively correct" 3-clock, mid-value select synchronization algorithm is not fault-tolerant. A single faulty clock can cause the other two good clocks to become desynchronized. The clock synchronization problem is far more subtle than it appears on the surface. Ad hoc algorithms are often assumed to be correct without rigorous analysis. Furthermore, despite the fact that the synchronization algorithm is the foundation of many fault-tolerant systems, the probability of system failure due to a synchronization failure traditionally has not been included in the reliability analysis of such systems (ref. 1.).

Recently, many provably correct fault-tolerant clock synchronization algorithms have appeared in the literature. Provided with each algorithm is a mathematical theorem which provides a bound on the clock skew as a function of measurable system parameters such as the maximum drift rate between good clocks or the maximum error in reading another non-failed processor's clock. It is no more difficult to build a system using one of these provably correct algorithms than it is to build one using an algorithm based on intuition. It is not necessary for fault-tolerant system designers to invent new algorithms and perform elaborate mathematical proofs, since such work is readily available. System designers can concentrate on methods to efficiently implement these existing algorithms. Some important implementation issues are discussed in Appendix A.

It is the goal of this paper to present (in a consistent notation) the fault-tolerant clock synchronization algorithms which have appeared in the literature. The associated performance theorems will be presented along with a discussion of the assumptions of the techniques.

SYMBOLS

b	mean time for a processor to read another processor's clock
$C_p(t)$	clock p's value at real time t
e_{qp}	clock read error -- the error in the clock value obtained by processor p when reading clock q
m	number of faulty clocks in the system
n	total number of clocks in the system
$r_p(T)$	the real time when clock p's value is T
R	time between resynchronizations (i.e. synchronization period)
S	time required to execute/perform synchronization algorithm
$T^{(i)}$	the i^{th} synchronization period
δ	maximum skew between any two clocks in the system
δ_0	initial skew between clocks in the system
Δ_{qp}	the apparent clock skew between processors q and p as perceived by processor p.
$\Delta_p^{(i)}$	the correction to clock p during the i^{th} resynchronization period
ε	the maximum clock read error (e_{qp})
ρ	maximum drift rate of all the clocks in the system

PRELIMINARY CONCEPTS

Definition of a clock

It is convenient to define a clock as a function from real time t to clock time T: $C(t) = T$. Real time will be distinguished from clock time by the use of small letters for real time and capital letters for clock time. The concept of a clock function is illustrated in figure 1. Since a properly functioning clock is a monotonic increasing function, its inverse function is well-defined. Some of the clock synchronization theorems are formulated using the clock function and others using the inverse function: $r(T) = C^{-1}(T) = t$. A clock function will be designated by a capital C and the inverse function by a lowercase r.[1]

---

1 Although $C_p(t)$ and $r_p(T)$ are different functions, they represent the same clock from two different perspectives. Sometimes it is necessary to switch from one view to the other. This does not change the clock itself.

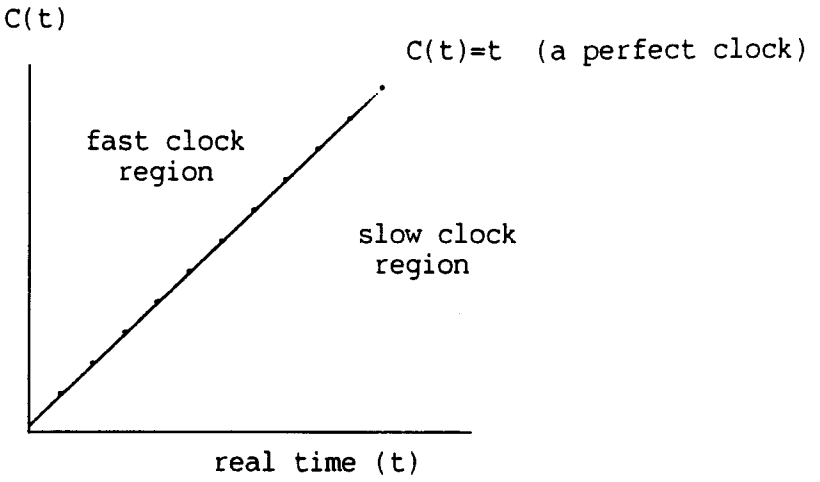

Figure 1. - The clock function

Subscripts will be used to distinguish different clocks, for example,

$C_p(t)$ -- clock p's value at real time t
$C_q(t)$ -- clock q's value at real time t
$r_p(T)$ -- the real time when clock p's value is T

We will let n represent the total number of clocks in the system and m the number of faulty clocks.

Drift Rates of Nonfaulty Clocks

A fault tolerant system typically consists of several processors each with its own local clock. Unfortunately, even when a processor is non-faulty its clock does not maintain perfect time. Therefore, it is necessary to define the concept of clock drift rate.

DEFINITION 1 A clock r(T) has an instantaneous drift rate $D(T) = |r'(T) - 1|$ at clock time T.

DEFINITION 2a A clock is nonfaulty if r(T) is a monotonic, differentiable function of T and there exists a ρ such that:

$$D(T) = | r'(T) - 1 | < \rho/2. \qquad (1)$$

Thus, the drift rate of a nonfaulty clock is bounded by $\rho/2$. A typical value of ρ is 1 μsec/sec or 10^{-6}. An alternate formula for ρ in terms of $r(T)$ is easily derived:

$$| r'(T) - 1 | < \rho/2 \tag{2}$$

$$-\rho/2 < r'(T) - 1 < \rho/2 \tag{3}$$

$$\int_{T_1}^{T_2} 1-\rho/2 \, dT < \int_{T_1}^{T_2} r'(T) \, dT < \int_{T_1}^{T_2} 1+\rho/2 \, dT \tag{4}$$

$$(1-\rho/2)(T_2-T_1) < r(T_2) - r(T_1) < (1+\rho/2)(T_2-T_1) \tag{5}$$

$$1-\rho/2 < \frac{r(T_2) - r(T_1)}{T_2 - T_1} < 1+\rho/2 \tag{6}$$

Letting $t_2 = r(T_2)$ and $t_1 = r(T_1)$, we can write $C(t_1) = T_1$ and $C(t_2) = T_2$. the above formula can then be rewritten as

$$\frac{1}{1+\rho/2} < \frac{C(t_2) - C(t_1)}{t_2 - t_1} < \frac{1}{1-\rho/2} \tag{7}$$

It is possible to start with this formula as the definition of clock drift. If this is done, the requirement of differentiability of the clock function can be removed.

Some synchronization algorithms are defined using the following alternative definitions of a good clock:

DEFINITION 2b A clock C is a nonfaulty clock if there exists a ρ_2 such that for all t_1, t_2:

$$1-\rho_2 < \frac{C(t_2) - C(t_1)}{t_2 - t_1} < 1+\rho_2 \tag{8}$$

DEFINITION 2c A clock C is a nonfaulty clock if there exists a ρ_3 such that for all t_1, t_2:

$$\frac{1}{(1+\rho_3)} < \frac{C(t_2) - C(t_1)}{t_2 - t_1} < 1+\rho_3 \tag{9}$$

The near equivalence of these definitions can be seen by examining the Taylor series expansion of $(1+\rho)^{-1}$:

$$(1+\rho)^{-1} = 1 - \rho + \rho^2 - \rho^3 + \rho^4 - \ldots \tag{10}$$

Thus for small ρ:

$$(1+\rho)^{-1} \simeq 1 - \rho \tag{11}$$

$$(1-\rho)^{-1} \simeq 1 + \rho \tag{12}$$

For the typical value of $\rho = 10^{-6}$ the difference between $(1-\rho)$ and $(1+\rho)^{-1}$ is on the order of 10^{-12}. Thus $\rho_2 \simeq \rho_3 \simeq \rho/2$.

Synchronization

DEFINITION 3 Two clocks r_p and r_q are synchronized to within δ of each other at clock time T if

$$|r_p(T) - r_q(T)| < \delta. \tag{13}$$

Since the clocks drift apart, it is necessary to periodically resynchronize the clocks of the system. There are two basic classes of synchronization algorithms: continuous-update and discrete-update. In the continuous update class, the frequency of the clock oscillator is continuously updated by analog circuitry. This method is used in phase-lock methods. In the discrete-update algorithms, the clock value and/or frequency is changed at discrete intervals. The time of resynchronization is typically determined by each processor from its own local clock. The period of time between resynchronizations is usually

constant, say R. Using $T^{(i)}$ as the clock time at the beginning of the i^{th} period, $T^{(i)} = T^{(0)} + iR$. For each period there is a different clock definition:

$$C_p^{(i+1)}(t) = C_p^{(i)}(t) + \Delta_p^{(i)} \tag{14}$$

where $\Delta_p^{(i)}$ is the i^{th} clock correction. Thus, the time base of the system is formally represented by a sequence of mathematical functions each applicable to a different interval of real time. This is illustrated in figure 2.

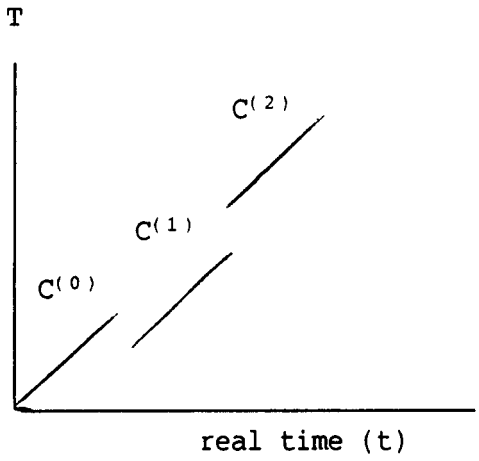

Figure 2. Sequence of clock functions

Each function differs from its predecessor by a constant. The time S required to execute the synchronization algorithm must be less than R.

Clock Read Error

In clock synchronization algorithms, it is necessary that a processor determine its clock skew with respect to every other clock in the system. This is logically equivalent to reading another processor's clock. Since the process of reading another processor's clock is subject to error, a processor can only obtain an approximate view of its skew with respect to other processors. The notation Δ_{qp} will be used to represent processor p's approximate view of its skew with respect to processor q. The following definition formalizes this concept:

DEFINITION 4 When processor p reads processor q's clock, processor p obtains an approximate skew Δ_{qp}. Processor p's error e_{qp} in reading processor q's clock is:

$$e_{qp}(T) = r_p^{(i)}(T + \Delta_{qp}) - r_q^{(i)}(T) \tag{15}$$

DEFINITION 5 If processors p and q are nonfaulty then processor p obtains an approximate skew Δ_{qp} such that

$$| e_{qp}(T) | < \varepsilon \tag{16}$$

Thus, ε is a bound on the clock read error and will be referred to as the maximum clock read error.

Usually the approximate skew Δ_{qp} is determined by reading another processor's clock. First, processor p reads q's clock at real time t_1 and obtains $C_q(t_1)$. If processor p subsequently reads its own clock at real time t_2 obtaining $C_p(t_2)$, then the approximate skew between the two processor's clocks can be calculated as follows:

$$\Delta_{qp} = C_p(t_2) - [C_q(t_1) + b] \tag{17}$$

where b is an implementation parameter. The value of b is the mean time for processor p to read processor q's clock.[2] In any real system, there will be some variation in the time to read another processor's clock. The error in Δ_{qp} is attributable to the variation in the time it takes to read the other processor's clock.

All clock synchronization algorithms discussed in this paper depend on the assumption of a bounded read error. Since the cause of the read error is the variation in the communication times between the processors, ε can also be shown to be a bound on the communication variation, i.e., the communication time is greater than b-ε and less than b+ε. This relationship is derived in the next section.

---

2 There may be many components of b -- time to read clock p, time to read clock q, time to transmit clock q to processor p, time for processor p to recognize receipt of clock q, etc.

Read error and communication delay. The relationship between clock read error and the communication-time variation is easily derived from the last two definitions. By substituting equation (17) in (15), we have

$$e_{qp}(T) = r_p^{(i)}(T + C_p(t_2) - [C_q(t_1) + b]) - r_q^{(i)}(T) \tag{18}$$

By definition 5 we have,

$$| r_p^{(i)}(T + C_p(t_2) - [C_q(t_1) + b]) - r_q^{(i)}(T) | < \varepsilon \tag{19}$$

Since the above formula is true for all T it is certainly true for $T=T_1$. Thus we have

$$| r_p^{(i)}(T_1 + C_p(t_2) - C_q(t_1) - b) - r_q^{(i)}(T_1) | < \varepsilon \tag{20}$$

Since $T_1 = C_q(t_1)$,

$$| r_p^{(i)}(C_p(t_2) - b) - r_q^{(i)}(T_1) | < \varepsilon \tag{21}$$

Using the highly accurate approximation in lemma 1 of the appendix B which is valid for small b,

$$| r_p^{(i)}(C_p(t_2)) - b - r_q^{(i)}(T_1) | < \varepsilon \tag{22}$$

Using $t_1 = r_q^{(i)}(T_1)$

$$| t_2 - t_1 - b | < \varepsilon \tag{23}$$

This yields

$$b - \varepsilon < t_2 - t_1 < b + \varepsilon \tag{24}$$

Thus, the communication delay $t_2 - t_1$ is bounded by $b \pm \varepsilon$ (i.e. the average delay $\pm$ max. clock read error).

Initial Synchronization

Many fault-tolerant clock synchronization algorithms are dependant on a close initial synchronization. Some authors argue that since the system initialization process is not critical (if it fails, you just start over), the initialization procedure does not have to be fault tolerant. Others either provide an alternate algorithm for initialization or explicitly provide for it in the main algorithm. If a fault-tolerant initialization procedure is not provided it is necessary to at least develop a technique to detect when the level of initial synchronization is not adequate.

LAMPORT, MELLIAR-SMITH INTERACTIVE CONVERGENCE ALGORITHM

The Lamport and Melliar-Smith algorithm (LMS) is based on a modified average. (See ref. 2.) This algorithm is in the class of discrete-update algorithms. When a processor's clock reaches the next resynchronization time, the processor first estimates its skew relative to every other processor in the system. All skews which are greater than a fixed value Ω are ignored. The mean of the remaining clock skews is used as the correction factor.

ALGORITHM: for all processors p

$$C_p^{(i+1)} = C_p^{(i)} + \Delta_p$$

where

$$\Delta_p = (1/n) \sum_{r=1}^{n} \overline{\Delta}_{rp}$$

if $r \neq p$ and $|\Delta_{rp}| < \Omega$ then $\overline{\Delta}_{rp} = \Delta_{rp}$

else $\overline{\Delta}_{rp} = 0$

Lamport and Melliar-Smith proved the following theorem which characterizes the worst-case performance of this algorithm in terms of low-level system parameters. (See ref. 2.)

THEOREM: If

$$3m < n$$

$$\delta \geq \text{Max} \left\{ \frac{n}{n - 3m} \left[2\varepsilon + \rho(R + 2 \frac{n - m}{n} S) \right] , \delta_0 + \rho R \right\}$$

$$\delta << \text{Min} \{ R, \varepsilon/\rho \}$$

Then the ALGORITHM satisfies the following:

S1. If up to time $T^{(i+1)}$ processors p and q are nonfaulty, then for all T in the interval $[T^{(i)}, T^{(i+1)}]$:

$$|r_p^{(i)}(T) - r_q^{(i)}(T)| < \delta$$

S2. If process p is nonfaulty up to time $T^{(i+1)}$, then

$$|r_p^{(i+1)}(T) - r_p^{(i)}(T)| < \Omega.$$

Statement (S1) of the theorem states that the maximum skew between any two nonfaulty clocks will be less than δ. The lower bound on δ is given in the theorem as a function of parameters n, m, ε, ρ, Ω, R, and S. Statement (S2) guarantees that the maximum correction will be less than Ω. The theorem is true for any value of δ which satisfies the above constraints. The constraint $\delta < \delta_0 + \rho R$ reveals that the algorithm cannot guarantee that synchronization will ever be any tighter than the initial degree of synchronization + the maximum amount of separation that can occur between two non-faulty clocks during an interval of length R.[3] The second constraint is

$$\delta \geq \frac{n}{n - 3m} \left[2\varepsilon + \rho(R + 2 \frac{n - m}{n} S) \right] \qquad (25)$$

---

3 The theorem is proved using a "worst case" analysis. Since the worst-case analysis includes the effect of malicious failure which is rare, the algorithm typically performs much better than the worst case result suggests. A stochastic analysis which determines an expected level of synchronization has not yet been performed. However, for life-critical applications, it is essential that all system functions must perform correctly in the presence of the worst-case behavior of the synchronization algorithm. This often results in a performance loss, but is unavoidable.

It is noteworthy that this bound on the clock skew is not only a function of clock read error ε but also a function of the number of active and faulty processors n and m respectively. Thus, the worst-case performance of the algorithm will vary when processors fail and when the system reconfigures. This can significantly complicate the reliability analysis of the system. (See ref. 1.)

This algorithm implicitly assumes that every processor can directly communicate with every other processor in the system.

LUNDELIUS AND LYNCH ALGORITHM

The Lundelius and Lynch algorithm (LL) runs periodically on each of the processors in the system (ref. 3) and thus is a discrete-update algorithm. When each processor's clock reaches $T^{(i)}$, it broadcasts a message. The algorithm is based on the assumption that each processor can directly communicate with every other processor. The processor collects clock messages from the other processors until $T^{(i)} + (1+\rho)(\delta+b+\varepsilon)$. These messages are used to create an ordered set of skews with respect to the other processors:

$$\{D_1, D_2, D_3, \ldots, D_k\} \qquad \text{where} \qquad n-m \leq k \leq n$$

and

$$D_1 < D_2 < D_3 < \ldots < D_k .$$

(Note. For each j there exists a q such that $D_j = \Delta_{qp}$ and for each p there exists an j such that $D_j = \Delta_{qp}$.)

ALGORITHM: For each processor p and a specific level of fault tolerance f, the following is performed:

$$C_p^{(i+1)} = C_p^{(i)} + \Delta_p$$

where

$$\Delta_p = (D_{1+f} + D_{n-f})/2$$

THEOREM: If

$$3m \leq 3f < n$$
$$R \geq 2(1+\rho)(\delta+\varepsilon) + (1+\rho)\max\{b, \delta+\varepsilon\} + \rho b$$
$$R \leq \delta/4\rho - \varepsilon/\rho - \rho(\delta+b+\varepsilon) - 2\delta - b - 2\varepsilon$$
$$\delta \geq 4\varepsilon + 4\rho(3\delta+b+3\varepsilon) + 8\rho^2(\delta+b+\varepsilon)$$

Then the ALGORITHM satisfies the following:

S1. If up to time $T^{(i+1)}$ processes p and q are nonfaulty, then for all T in the interval $[T^{(i)}, T^{(i+1)}]$:

$$|r_p^{(i)}(T) - r_q^{(i)}(T)| < \delta$$

S2. If process p is nonfaulty up to time $T^{(i+1)}$, then

$$\Delta_p < (1+\rho)(\delta+\varepsilon) + \rho b.$$

The theorem reveals that the algorithm will maintain synchronization to within δ as long as the specified constraints are met. The first constraint $3m < 3f < n$ shows that the algorithm can only tolerate a certain number of failures. The last 3 formulas can be used to determine the level of synchronization obtainable by the algorithm. For a particular system, the parameters ρ, ε, and b are fixed. Therefore, these formulas define the relationship between δ and R for a particular system. The first of these three formulas constrains the synchronization period. Ignoring terms containing ρ (since it is usually very small -- 10^{-6}) this formula becomes:

$$R \geq 2(\delta+\varepsilon) + \max\{b, \delta+\varepsilon\} \qquad (26)$$

Since b, δ, ε are typically on the order of 10^{-3} seconds and R is on the order of 10^{-1}, this constraint is usually insignificant. The last two equations impose a lower bound on δ. By simple algebraic manipulation these equations can be written as:

$$\delta \geq \frac{4\rho R + 4\varepsilon(1+2\rho+\rho^2) + 4\rho(1+\rho)b}{1 - 8\rho - 4\rho^2} \tag{27}$$

$$\delta \leq \frac{4\varepsilon + 4\rho(b+3\varepsilon) + 8\rho^2(b+\varepsilon)}{1 - 12\rho - 8\rho^2} \tag{28}$$

Once again, since ρ is a dimensionless quantity which is very small, these equations can be rewritten as:

$$\delta \leq 4\rho R + 4\varepsilon + 4\rho b \tag{29}$$

$$\delta \leq 4\varepsilon + 4\rho(b+\varepsilon) \tag{30}$$

The above two constraints are satisfied whenever:

$$\delta \leq 4\varepsilon + 4\rho(R+b+\varepsilon) \tag{31}$$

This formula can be used as the maximum clock skew which will occur in a system which uses this algorithm. The theorem was proved using definition 2c for drift rates and assuming direct communication between the processors of the system.

HALPERN, SIMMONS, STRONG, DOLEV ALGORITHM

The Halpern, Simmons, Strong, Dolev (HSSD) algorithm is a discrete-update algorithm, but differs from the LMS and LL algorithms in that it does not rely on some form of averaging. (See ref. 4.) Periodically each processor seeks to be the synchronizer of the system. The non-faulty processors each know what time the next synchronization interval will occur. They all seek to become the synchronizer at approximately the same time. When all of the processors are not faulty, only one of the processors becomes the synchronizer. If the synchronizer fails, the algorithm is designed such that the remaining good processors effectively take over and synchronize despite the erroneous behavior of the synchronizer.

The algorithm relies on the use of unforgeable digital signatures. It is necessary that each processor be able to encode a message using a unique encoding function such that no other processor can generate or alter the message without invalidating it. Standard techniques exist for implementing digital signatures. (See ref. 5.) Furthermore, every processor can determine who encoded the message. Thus, a processor can determine if a message is authentic or forged.

The bound on the clock drift rate is defined using definition 2c. The clock read error is not directly specified. The theorem which characterizes the algorithm was proved under the assumption that the communication delay is bounded by 0 and an upper bound (i.e. $b+\varepsilon$). This is less restrictive than is implied by definition 2. Thus, the theorem should still be true if the clock read error were defined in terms of definition 2.

Each processor p executes the following two concurrent tasks:

```
Task TM:
   IF C_p(t) = ET THEN
      SIGN_AND_SEND "The time is ET";
      C_p(t) := ET;
      ET := ET + R;
   ENDIF;

Task MSG:
   IF an authentic message  M  is received saying "The time is T" THEN
      S := the number of distinct signatures
      IF T=ET and ET-S*D < C_p(t) THEN
         SIGN_AND_SEND M;
         C_p(t) := ET;  ET := ET + R;
      ENDIF
   ENDIF
```

These tasks execute as concurrent processes on a processor. If an authentic message is received on a processor before its clock reaches the next synchronization time ET, then task MSG is executed. If no authentic message has been received and a processor's clock reaches ET then task TM is executed. Note, that after TM sends a message, ET is incremented. Thus, after TM is complete all the other synchronization messages for this period are ignored.

Likewise, after task MSG signs a message, ET is incremented so that TM cannot send a message until the next synchronization period. A key aspect of this algorithm is the ET-S*D < $C_p(t)$ test of task MSG. The "window" of acceptance is a function of the number of signatures of the message. If there is one signature, then the message must arrive in the interval [ET-D,ET], in order for the message to be signed and forwarded. If there are 3 signatures, then the message must arrive in the interval [ET-3*D,ET]. It is assumed that these tasks are non-preemptable.

THEOREM If

$$\delta = (1+\rho)S + \rho(2+\rho)R$$
$$D >= \delta$$
$$R \geq S(1+\rho) + mD$$

Then

S1: $| C_i(t) - C_j(t) | < \delta$

S2: $| C_i^{(k)}(t) - C_i^{(k+1)}(t) | < (m+1)D$

This algorithm does not require direct communication between all of the processors of the system. In fact as long as a good processor is in contact with another good processor they remain synchronized. The key parameters in this algorithm are S and R. The parameter S represents the maximum time required to execute the synchronization algorithm. In the proof of the theorem, this is shown to be the maximum delay in sending a message from a good processor to another good processor in the network. This is obviously a function of the structure of the network. The theorem states that as long as the good processors can communicate, they will remain synchronized to within δ. But since δ is a function of S and S is a function of network connectivity, the level of synchronization obtainable may degrade as connections are lost in the network. The parameter R is the period of resynchronization. As expected, increasing the frequency of resynchronization increases the degree of synchronization obtained.

Halpern, Simons, Strong and Dolev also presented a more powerful form of their algorithm which can synchronize an unsynchronized clock. This extended

algorithm is capable of bringing a repaired processor back into synchronization with the rest of the processors. This algorithm extends the basic algorithm with a Byzantine broadcast among the active processors in order to agree when to allow a processor to join them. The details of this algorithm will not be presented here, but can be found in reference 4.

KESSEL'S ALGORITHM

In this section one of the two algorithms developed by J. L. W. Kessels is presented. (See ref. 6.) The other algorithm is basically the same as the one discussed in this section, except that it utilizes an analog circuit which was not characterized formally. No proof of correctness was given for this algorithm. Kessel's synchronization method is a hardware solution and falls in the class of continuous-update algorithms. It depends fundamentally on the assumption that the transmission delays between the separate clocks are negligible. If there is a significant variation in these delays, the algorithm will not work. Thus, this algorithm assumes that $\varepsilon \approx 0$.

A clock is defined in terms of signal that periodically transitions between two states. The states of this clock are numbered sequentially. The value of the clock is the number of the last transition. Two clocks are defined to be synchronized if their values are equal for at least some part of their state interval. If the minimum frequency of the clocks is υ, the maximum clock skew is $1/(2\upsilon)$.

Kessels first presents his algorithm in terms of a circuit block diagram. He later demonstrates that it is equivalent to a concurrent program whose correctness can be formally analyzed. In this section only the concurrent algorithm will be presented. For details on an appropriate hardware solution and details about the correctness proof the reader is referred to reference 6.

The notation Nj: P(j) will be used as an abbreviation for the number of values of j for which P(j) is true. For example

$$Nj: C_i(t) > C_j(t)$$

is an abbreviation for

$$| \{ j \mid C_i(t) > C_j(t) \} |$$

which is the number of clocks which are greater than clock j.

The concurrent algorithm is described using Dijkstra's concept of guarded commands. (See ref. 7.) Each command separated by | is a guarded command. The guarded command consists of a guard (i.e. a Boolean expression) followed by a → and a statement. All of the guarded commands within the **do** - **od** loop, which are separated by | execute concurrently. As long as one of the Boolean expressions is true the loop is continued. If more than one guard is currently true then one of them is selected nondeterministically (i.e. which one is selected is determined randomly.)

The Kessels algorithm consists of n concurrent processes. The algorithm for the i^{th} process (on processor i) is:

do

$[N_j : C_i(t) > C_j(t)] > f$ → **skip**

| $[N_j : C_i(t) > C_j(t)] \leq f$
and $[N_j : C_i(t) < C_j(t)] \leq f$
and $k_i < K - 1$ → $k_i := k_i + 1$

| $[N_j : C_i(t) < C_j(t)] > f$ **or** $k_i = K - 1$ → $k_i := 0;\ C_i(t) := C_i(t) + 1$

od

Since it is impossible that all of the guards become false simultaneously, the loop never terminates. The correctness of the above algorithm depends on the ability of a good clock to read all of the other clocks with negligible error. Exactly what constitutes a negligible read error is never defined. Furthermore, there is no mathematical proof which relates the impact of read error on the synchronization. The parameter K determines the rate at which the clock counter is incremented. The clock rate is K times the time required to execute the loop. Kessels shows how a circuit can be designed to insure that all the good processors compare their values with the other clocks when none of the clocks are in transition. He refers to this as the "interlude" phase.

ALGORITHMS BASED ON PHASE-LOCKING

Given two non-faulty voltage-controlled oscillators, phase-locked loop circuitry can be used to keep them synchronized. For more details on such circuitry the reader is referred to ref. 8. In this section fault-tolerant algorithms which use this phase locking technique will be discussed. Each of these algorithms depend on the assumption that the phase-locking circuitry maintains adequate synchronization between two non-faulty oscillators. The basic problem is how to select a "standard" signal for each clock in the system in a manner that guarantees that all non-faulty clocks will remain synchronized despite the arbitrary behavior of the faulty clocks.

T. Basil Smith designed a fault-tolerant four-clock, phase-locking synchronization algorithm for the Fault-Tolerant Multi-Processor (FTMP). (See refs. 8 and 9.) Each clock in the system observes the outputs of the other three clocks continuously and determines its phase difference with respect to each of them. These phase differences are ordered:: $T_1 \leq T_2 \leq T_3$. Each clock selects the second signal as the reference signal to which it can phase-lock. Phase-lock algorithms are continuous-update algorithms.

The degree of synchronization obtained is dependent on the effectiveness of the phase-lock circuitry. This circuitry is intrinsically analog and belongs to the class of continuous-update algorithms. The developers of these clock synchronization algorithms have not included a mathematical analysis of the phase-lock circuitry used in their algorithm. This is a distinct disadvantage of the phase-clock algorithms -- the maximum clock skew has not been characterized in terms of the specific parameters of the phase-lock circuitry.

KRISHNA, SHIN, BUTLER ALGORITHM

The algorithm presented by Krishna, Shin, and Butler (see ref. 10) is a generalization of Smith's algorithm for more than four clocks. Surprisingly, the median signal does not work for higher levels of redundancy. With a median select algorithm, it is possible for malicious failures to partition the set of clocks into two or more separate "cliques" which are internally synchronized but not synchronized with other cliques. To see this, suppose we have a system of 7 clocks. Such a system should be able to mask the failure of

two clocks. Suppose that the 5 good clocks are named a, b, c, d, e and the bad clocks are named x and y. If the transmission delays between clocks are negligible, then the order of arrival of all the signals from the good clocks should be the same on all the processors. If the bad processors fail maliciously, their order may be seen differently by different processors. Consider the following ordering of signals seen by each of the processors in the system.

order seen by a: x y a b c d e
order seen by b: x y a b c d e
order seen by c: a b c d e x y
order seen by d: a b c d e x y
order seen by e: a b c d e x y

The order of the signals from the good processors a,b,c,d and e is seen consistently by all the processors. The faulty processors x and y are seen differently by the processors of the system. Letting $c \leftarrow b$ represent the relation that b synchronizes to c, If each processor selects the median signal (i.e. the third signal; underlined above) not including itself, then the following synchronizations will occur from the above scenario:

$$a \leftrightarrow b \qquad\qquad e \rightarrow c \leftrightarrow d$$

Thus {a,b} and {c,d,e} form non-synchronizing cliques.

The following algorithm has been shown to prevent the creation of non-synchronized cliques.

THEOREM If $n \geq 3m+1$ and the signal selected on processor p, $f_p(N,m)$, is defined as follows

$$f_p(n,m) = \begin{cases} 2m & \text{if } A_p < n-m \\ m+1 & \text{if } A_p \geq n-m \end{cases}$$

where A_p is the order of processor p in the temporal sequence of arriving clock signals, then all the non-faulty clocks of the system will synchronize.

For the situation above n=7, m=2:

$$f_p(N,m) = \begin{cases} 4 & \text{if } A_p < 5 \\ 3 & \text{if } A_p \geq 5 \end{cases}$$

thus

$f_a = 4$ and $a \rightarrow b$
$f_b = 4$ and $b \rightarrow c$
$f_c = 4$ and $c \rightarrow e$
$f_d = 4$ and $d \rightarrow e$
$f_e = 3$ and $e \rightarrow c$

The following synchronizing relationship results with no cliques:

$$a \rightarrow c \leftrightarrow e$$
$$\uparrow \quad \uparrow$$
$$b \quad d$$

This algorithm depends on the use of phase-lock circuitry which was not characterized formally. No theorem was presented which relates maximum clock skew to parameters of the phase-lock circuitry.

CONCLUDING REMARKS

The synchronization of the clocks of a multi-processor system is a critical function in a fault-tolerant system. Regardless of the level of redundancy in the system, if the clocks become deskewed, total system failure is almost always certain. It is imperative that provably correct algorithms be used in the design of a fault-tolerant computer system. Many such algorithms have appeared in the engineering literature. Unfortunately, these algorithms and their associated performance theorems are difficult to decipher and compare, since they are presented in different notations. This paper presents in the same notation six different algorithms which have appeared in the literature. It is hoped that future critical system developments will take advantage of these algorithms whose performance properties have been analyzed mathematically. The provably correct algorithms are no more difficult to implement than ad-hoc techniques. It is suggested that future system designers either exploit the available algorithms and concentrate on efficient and correct implementation or mathematically prove the algorithms which they develop for their systems.

APPENDIX A

IMPLEMENTATION OF SYNCHRONIZATION ALGORITHMS

There are three basic problems to be solved when implementing a clock synchronization algorithm:

(1) scheduling the synchronization algorithm on the local processors

(2) reading all the other processor's clocks in the system

(3) computing a correction based upon the algorithm and updating the local clock

In the next two sections these steps will be discussed.

Step 1 - Scheduling the Synchronization Task

The clock synchronization algorithms discussed in this paper depend upon periodic execution. Most real-time systems are based on a cyclic scheduler, so the clock synchronization algorithms fit naturally into such a system. These systems are driven by a clock-interrupt. The interrupt is set to fire periodically. When the interrupt fires, the processor immediately transfers control to the scheduler which then transfers control to a task. This can usually be accomplished with only a minimum overhead. The scheduling requirements of the clock synchronization task can usually be met with a high degree of accuracy.

Step 2 - Reading Clocks

There are two basic approaches to implementing the clock read function in a system of multiple processors -- (1) a distributed read (2) coordinated broadcast.

METHOD 1 - distributed read. - If the system is designed such that a processor can read the clock of another processor (say be direct memory access), then the read error becomes a function of the worst case memory contention time. The clock could be a multi-ported memory-mapped i/o device. Conceptually, this represents the simplest method of implementing the distributed clock read function. It is essential that the global read be implemented so that a processor can only read a portion of another processor's memory. In this way, another processor cannot interfere with the activities of the local processor by contending for its memory. The degree of synchronization obtainable, will depend on the magnitude and variation in the times required to read another processor's clock. Direct bi-directional point-to-point access will provide the most accurate clock-reading capability.

METHOD 2 - coordinated broadcast. - If a system of processors has a broadcast capability, a global clock read capability can be implemented using this facility. This was done for the SIFT computer. (See ref. 1.) It should be noted that when this is done, the clock read function depends on the synchronization algorithm.[4] The beginning of the synchronization task is divided into a sequence of "windows", one window per processor. During its transmit window, a processor repeatedly reads its clock and broadcasts its value. All other processors wait until either the clock value arrives or the window ends. Since the accuracy of processor p's perception of clock q depends on how quickly processor p recognizes receipt of clock q's value, processor p executes a tight "wait-for loop" until clock q's value arrives. When the clock value is received, processor p reads its clock. The skew is calculated as the difference between the two clocks minus the approximate communication delay. When all windows are completed, the correction is calculated, and the clock is corrected. Within its broadcast window, a processor q reads its clock at real time t_1 and transmits the value $C_q(t_1)$ to process p. Upon receiving the message at t_2, processor p immediately reads its clock to obtain $C_p(t_2)$. This is illustrated in figure 3.

---

4 Thus, there is a mutual dependency -- the synchronization algorithm depends on the clock read function and the clock read function depends on the synchronization. If this technique is used, then the correctness proof of the synchronization system must deal with the clock read method and synchronization algorithm simultaneously. The proofs cannot be decoupled.

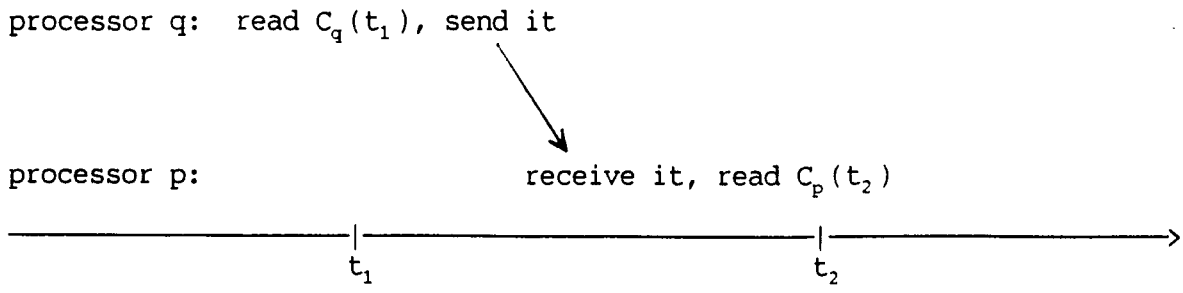

Figure 3. - reading another processor's clock

If the exact communication delay B_{qp} (i.e. $C_q(t_2) - C_q(t_1)$) were known, then the exact skew A_{qp} at real time t_2 could be calculated by

$$A_{qp}(t_2) = C_p(t_2) - C_q(t_1) - B_{qp}.$$

Since the communication delay is variable, B_{qp} is a random variable and is never exactly known by the synchronization algorithm. Thus, the designer of the synchronization system chooses a value b approximately equal to $E(B_{qp})$ which is used to compute an apparent skew Δ_{qp} by the following formula:

$$\Delta_{qp}(t_2) = C_p(t_2) - C_q(t_1) - b.$$

These apparent skews are used to calculate the clock correction value Δ_p according to the synchronization algorithm.

The apparent skew Δ_{qp} differs from the actual skew A_{qp} by an error $e_{qp} = \Delta_{qp} - A_{qp} = B_{qp} - b$. There are two components to this error:

$$e_{qp} = B_{qp} - b = [B_{qp} - E(B_{qp})] + \mu,$$

where $\mu = E(B_{qp}) - b$. The first component, $B_{qp} - E(B_{qp})$, is the variation due to the random nature of the communication. The second component, μ, is a bias due to the error in choosing b. Also, it follows from the above formula that

$$E(e_{qp}) = \mu.$$

The performance of the theoretical algorithms are typically specified by theorems which are expressed in terms of ε, a theoretical upper bound on e_{qp}. Unfortunately, e_{qp} is defined in terms of real time rather than observable clock time. The formula

$$C_p(t) - C_q(t) + e_{qp} \simeq \Delta_{qp}$$

which relates the theoretical e_{qp} to observable clock values is shown to be a highly accurate approximation in appendix A.

The major source of read error in the broadcast method is the busy-wait in the CPU. If the looping time of the software which waits for the arrival of a broadcasted clock value is w milliseconds, the maximum read error is at least w milliseconds. If special circuitry is designed to recognize the arrival of a clock value and immediately latch the current value of its own clock this can decrease the read error considerably.

The concept of sending a specific processor a message is essentially identical to the broadcast method. However, message facilities are often implemented with hardware interrupts. It is very dangerous to allow a processor to interrupt another processor in a fault-tolerant computer. Message systems for a critical system are best implemented by periodically having the receiving processors examine their mail-boxes.

Step 3 - Correcting Local Clocks

Once the values Δ_{qp} have been determined by a processor p, the fault-tolerant clock synchronization algorithms require the determination of a correction factor Δ. These calculations are usually very simple, e.g. a mean or median. Next, the processor must correct its local clock. Note, that it is not enough to merely maintain a "correction value" in local memory. It is necessary that the internal state of the clock which fires the clock-interrupt be changed, since this interrupt triggers the periodic execution of the synchronization task. In software-implemented algorithms, a special instruction must be provided by the processor to enable program-level modification of the clock.

APPENDIX B

USEFUL APPROXIMATIONS INVOLVING CLOCK FUNCTIONS

Lemma 1 If clock r is non-faulty and Δ is small, then $r(T_1+\Delta) \simeq r(T_1) + \Delta$.

From the definition of ρ we have

$$1-\rho/2 < \frac{r(T_2) - r(T_1)}{T_2 - T_1} < 1+\rho/2$$

Letting $\Delta = T_2 - T_1$:

$$|r(T_1+\Delta) - r(T_1) - \Delta| < (\rho/2)\Delta$$

Since ρ is typically on the order of 10^{-6} and Δ is usually much less than 10^{-2}, and r is typically on the order of 10^{-2}, we have

$$\text{rel. error} = \frac{|r(T_1+\Delta) - r(T_1) - \Delta|}{r(T_1)} < \frac{(\rho/2)\Delta}{r(T_1)} \leq \frac{10^{-8}}{10^{-2}} = 10^{-6}$$

Thus, $r(T_1+\Delta) \simeq r(T_1) + \Delta$.

Lemma 2 If clock C is non-faulty and Δ is small, then $C(y+\Delta) \simeq C(y) + \Delta$

Proof: Let $x = r(T+\Delta)$ and $y = r(T)$. Then $C(x) = T + \Delta$ and $C(y) = T$ since C is the inverse function of r. It directly follows that $C(x) = C(y) + \Delta$. By Lemma 1, we have: $x \simeq y + \Delta$. Hence, $C(x) \simeq C(y+\Delta)$. Substituting $C(y) + \Delta$ for $C(x)$, we have $C(y+\Delta) \simeq C(y)+\Delta$.

Theorem $C_p(t) - C_q(t) + e_{qp} \simeq \Delta_{qp}$.

proof: By definition $e_{qp} = r_p(T+\Delta_{qp}) - r_q(T)$. Letting $x = r_p(T+\Delta_{qp})$ and $t = r_q(T)$ implies $C_p(x) = T + \Delta_{qp}$ and $C_q(t) = T$. Thus, $C_p(x) = C_q(t) + \Delta_{qp}$. Since by the definition $e_{qp} = x - t$, we have $C_p(e_{qp}+t) = C_q(t) + \Delta_{qp}$. From lemma 2 we conclude that $e_{qp} + C_p(t) = C_q(t) + \Delta_{qp}$.

REFERENCES

1. Butler, Ricky W.; Palumbo, Daniel L.; and Johnson, Sally C.: Application of a Clock Synchronization Validation Methodology to the SIFT Computer System, IEEE Fifteenth International Symposium on Fault-Tolerant Computing (FTCS-15), June 19-21, 1985

2. Lamport, Leslie; and Melliar-Smith, P. M.: Synchronizing Clocks in the Presence of Faults, Journal of the ACM, Vol. 32, No 1., January 1985.

3. Lundelius, Jennifer; and Lynch, Nancy: A New Fault-Tolerant Algorithm for Clock Synchronization, ACM Conference on Principles of Distributed Computing, 1984.

4. Halpern, Joseph Y; Simmons, Barbara; Strong, Ray and Dolev, Danny: Fault-Tolerant Clock Synchronization, ACM Conference on Principles of Distributed Computing, 1984.

5. Rivest, R. L.; Shamir, A; and Adleman, L: A Method for Obtaining Digital Signatures and Public-key Cryptosystems, Communications of the ACM, Vol. 21, No. 2, 1978, pp.120-126.

6. Kessels, J. L. W.: Two Designs of a Fault-Tolerant Clocking System, IEEE Transactions on Computers, Vol. C-33, No. 10, October 1984.

7. Dijkstra, Edsger W.: A Discipline of Programming, Prentice-Hall, Inc. 1976.

8. Smith, T. B.: Fault-Tolerant Clocking System, IEEE Eleventh International Symposium on Fault-Tolerant Computing (FTCS-11), 1981, pp. 262-264.

9. Hopkins, A. L., et. al.: FTMP -- A highly reliable fault-tolerant multiprocessor for Aircraft, Proceedings of the IEEE, Vol. 66, pp. 1221-1239, Oct. 1978.

10. Krishna, C. M.; Shin, Kang G.; and Butler, Ricky W.: Ensuring Fault Tolerance of Phase-Locked Clocks, IEEE Transactions on Computers, Vol. c-34, No. 8, August 1985.

NASA National Aeronautics and Space Administration

Report Documentation Page

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
NASA TM-100553		

4. Title and Subtitle	5. Report Date
A Survey of Probably Correct Fault-Tolerant Clock Synchronization Techniques	February 1988
	6. Performing Organization Code

7. Author(s)	8. Performing Organization Report No.
Ricky W. Butler	
	10. Work Unit No.
	505-66-21-01

9. Performing Organization Name and Address	11. Contract or Grant No.
NASA Langley Research Center Hampton, VA 23665-5225	
	13. Type of Report and Period Covered
12. Sponsoring Agency Name and Address	Technical Memorandum
National Aeronautics and Space Administration Washington, DC 20546-0001	14. Sponsoring Agency Code

15. Supplementary Notes

16. Abstract

This paper examines six provably correct fault-tolerant clock synchronization algorithms. These algorithms are all presented in the same notation to enable easier comprehension and comparison. The advantages and disadvantages of the different techniques are examined and issues related to the implementation of these algorithms are discussed. The paper argues for the use of such algorithms in life-critical applications.

17. Key Words (Suggested by Author(s))	18. Distribution Statement
Clock synchronization Fault tolerant Formal verification Design proof	Unclassified - Unlimited Star Category 62

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of pages	22. Price
Unclassified	Unclassified	27	A03